Chapitre 3

Dans ce chapitre, nous allons enfin commencer à programmer !
Nous commencerons par une petite discussion sur le sujet "Kernel vs Nostub", puis nous réaliserons un premier programme, ne faisant rien, mais qui nous permettra de nous familiariser avec TIGCC, afin de pouvoir continuer sur des projets plus importants.

I:\ "Kernel vs Nostub" : Que choisir ?

Sur nos TIs, il existe deux modes de programmation. Le premier (historiquement parlant, puisqu'il a vu le jour au temps de la TI-92) est le mode Kernel. Le second, qui s'est répandu à peu près en même temps que la possibilité de coder en langage C pour nos TIs est le mode Nostub.
Chaque mode, quoi qu'en disent certains, présente des avantages, et des inconvénients. Je vais ici tenter de vous décrire les plus importants, afin que vous puissiez, par la suite, faire votre choix entre ces deux modes, selon vos propres goûts, mais aussi (et surtout !) selon ce dont vous aurez besoin pour votre programme.

Mode Kernel :

Avantages	Inconvénients
● Permet une utilisation simple de librairies dynamiques (équivalent des Dll sous Windows) déjà existantes (telles Ziplib, Graphlib, Genlib, ...), ou que vous pourriez créer par vous-même. ● Le Kernel propose de nombreuses fonctionnalités destinées à vous faciliter la vie, ainsi qu'un système d'anticrash parfois fort utile. (Une fois le kernel installé, l'anticrash l'est aussi, pour tous les programmes que vous exécutez sur la machine ; pas uniquement le votre !)	● Nécessite un programme (le Kernel) installé avant que vous ne puissiez lancer le votre. ● L'utilisation de librairies dynamiques fait perdre de la RAM lors de l'exécution du programme (parfois en quantité non négligeable) si toutes les fonctions de celle-ci ne sont pas utilisées. Cependant, notez qu'il est tout à fait possible de programmer en mode Kernel sans utiliser de librairies dynamiques ! Naturellement, la mémoire RAM est récupérée une fois l'exécution du programme terminée.

Mode Nostub :

Avantages	Inconvénients
● Ne nécessite pas de Kernel installé (Fonctionne même, normalement, sur une TI "vierge" de tout autre programme). ● En théorie, si le programme n'a pas besoin des fonctionnalités proposées par les Kernels (qu'il lui faudrait ré-implémenter !), il pourra être plus petit que s'il avait été développé en mode Kernel (car les programmes en mode Kernel sont dotés d'un en-tête de taille variable, qui peut monter à une bonne cinquantaine d'octets, et jamais descendre en dessous de environ 20-30 octets) Cela dit, en pratique, c'est loin de toujours être le cas, en particulier pour les programmes de taille assez importante.	● Ne permet pas, en ASM, la création et l'utilisation de librairies dynamiques (du moins, pas de façon aussi simple qu'en mode Kernel !) ; cela est actuellement permis en C, mais pas encore en ASM. ● En cas de modifications majeures (par Texas Instrument) dans les futures versions d'AMS, certaines fonctions d'un programme Nostub peuvent se révéler inaccessibles, et alors entraîner des incompatibilités entre la calculatrice et le programme. Il faudra alors que l'auteur du programme corrige son code et redistribue la nouvelle version du programme (Sachant que la plupart des programmeurs sur TI sont des étudiants, qui stoppent le développement sur ces machines une fois leurs études finies, ce n'est que rarement effectué !). Ce n'est pas le cas en mode Kernel, pour les fonctions des librairies dynamiques : l'utilisateur du programme n'aura qu'à utiliser un Kernel à jour pour que le programme fonctionne de nouveau.

Dans ce tutorial, nous travaillerons en mode Nostub. Non que je n'apprécie pas le mode Kernel (bien au contraire), mais le mode Nostub est actuellement le plus "à la mode". Je me dois donc presque dans l'obligation de vous former à ce qui est le plus utilisé...
Cela dit, il est fort probable que, dans quelques temps,nous étudions pendant quelques chapitres le mode Kernel, ceci non seulement à cause de son importance historique, mais pour certaines des fonctionnalités qu'il propose. A ce moment là, nous le signalerons explicitement.
Bien que n'étudiant pas tout de suite le mode Kernel, je tiens à préciser, pour ceux qui liraient ce tutorial sans trop savoir quel Kernel installer sur leur machine (s'ils souhaitent en installer un, bien entendu), que le meilleur Kernel est actuellement PreOS, disponible sur www.timetoteam.fr.st. C'est le seul qui soit à jour : DoorsOS est totalement dépassé, TeOS l'est encore plus, de même que PlusShell, et UniversalOS n'est plus à jour. De plus, PreOS propose nettement plus de fonctionnalité que DoorsOS ou UniversalOS ! (Notons que PreOS permet naturellement d'exécuter les programmes conçus à l'origine pour DoorsOS ou UniversalOS).

II:\ Notre tout premier programme

Non, Non, vous ne rêvez pas, nous allons enfin voir notre premier programme ! Il ne fera absolument rien, si ce n'est, une fois lancé, retourner le contrôle au système d'exploitation de la TI (AMS, aussi appelé TIOS (TI Operating System)), mais nous permettra de découvrir, en douceur, les bases indispensables à tout programme que nous écrirons par la suite en Assembleur.

A: Créer les fichiers nécessaires à notre programme

Nous supposerons que vous avez déjà installé le pack TIGCC, et ce de façon correcte.
Créez, quelque part sur votre disque dur, un dossier vide dans lequel nous placerons tous les fichiers de notre projet. Utiliser un dossier par projet est une habitude que je vous recommande vivement de prendre : même si, pour l'instant, nos projets ne se composeront pas d'une grande quantité de fichiers, d'ici quelques temps, lorsque vous travaillerez sur de gros projets, vous risquez d'être amené à utiliser beaucoup de fichier différents. Prenez donc l'habitude de bien les classer ! (Croyez-moi, quand on travaille sur un projet de plus de 500, voire même de plus de 1000, fichiers tout compris (codes sources, aide, images, archives externes, librairies et autres, on est bien content de les avoir bien ordonné dès le début (je ne plaisante pas !))
Nous désignerons, dans la suite de cette partie, ce dossier sous le nom "dossier1" ; libre à vous de choisir un nom plus évocateur !
Ouvrez le programme TIGCC IDE (Selon l'installation que vous avez effectué, un raccourci a normalement du être créé dans le menu démarrer de Windows).
Créez un nouveau projet ; pour cela, cliquez sur "File", puis "New", et enfin sur "Project". Nommez le du nom que vous souhaitez donner à votre programme. Dans la suite de nos explications, nous supposerons que vous avez choisi le nom "prog1".
A présent, il va falloir créer un fichier comportant l'extension ".asm", qui contiendra la texte de notre programme (son "code source"). Pour cela, cliquez sur "File", puis "New", puis sur "A68k Assembly Source File". Nommez ce fichier comme bon vous semble... (En général, je donne à mon fichier source principal le même nom qu'au projet, soit, ici, "prog1").
TIGCC IDE va alors vous créer un fichier, qui ne contiendra que ceci :

```
; Assembly Source File
; Created 11/02/2003, 23:18:07

    section ".data"
```

Il va à présent falloir taper notre code source...

B: Ce qu'il faut absolument mettre

Avant tout, une petite remarque : Il est possible d'insérer, dans le texte de notre programme, des commentaires. Un commentaire est un texte, qui est généralement destiné à faciliter la compréhension du source, et qui n'est pas pris en compte par le logiciel Assembleur.
En A68k, les commentaires doivent être précédés d'un point-virgule, qui signifie que tout ce qui est écrit après, jusqu'à la fin de la ligne, est un commentaire.
Sous TIGCC IDE, avec les paramètres par défaut, les commentaires sont écrits en vert.

Précisons une petite chose supplémentaire tout de suite : en Assembleur, la majorité des instructions doit être précédée d'au moins un espace (en règle générale, nous utiliserons un caractère de tabulation). En fait, seules les étiquettes ne sont pas précédées d'espace(s). Petit "truc" simple pour repérer les étiquettes : elles sont assez souvent suivie d'un caractère ":" (Bien que celui-ci soit facultatif, nous le noterons toujours, en particulier au début de ce tutorial, par habitude, et par soucis de clarté).

A présent, voici comment commencer l'écriture de notre code source :

Tout d'abord, puisque nous allons utiliser des données (déclarations de fonctions, constantes, et autres) "standard", il nous faut les inclure. Pour cela, il faut écrire cette ligne dans notre fichier source :

```
include "OS.h"
```

Ensuite, puisque nous avons décidé de programmer en mode Nostub, il faut dire au logiciel Assembleur que c'est le cas, afin de celui-ci sache quel type d'exécutable générer. Pour cela, il convient de rajouter à notre fichier source :

```
xdef _nostub
```

A présent, il faut préciser pour quelle calculatrice vous voulez programmer.
Si vous destinez votre programme à une TI-89, écrivez :

```
xdef _ti89
```

Si c'est pour TI-92+, ou alors pour V200, que vous rédigez votre programme, écrivez :

```
xdef _ti92plus
```

Bien entendu, vous pouvez mettre les deux, dans le cas où vous destineriez votre programme aux deux machines simultanément. C'est ce que nous ferons généralement dans nos exemples, du moins quand ils n'utiliserons pas de fonctionnalités, telles le clavier ou l'écran, de façon spécifique à l'un ou l'autre des modèles.

En mode Nostub, l'exécution du programme débute au début du code source que nous écrivons. Donc, le code que nous écrirons "en haut" du fichier source sera le premier exécuté. Cependant, par pure habitude, plus pour faire joli et faciliter le repérage qu'autre chose, nous placerons toujours notre code après un label (une étiquette) nommée "_main", que nous ajouterons ainsi à notre source :

```
_main:
```

Après ce label, nous pouvons écrire le code du programme. Notre premier, celui-ci, ne fera absolument rien. Nous n'avons donc pas énormément de code à rajouter.

Cependant, puisque le TIOS (le Système d'exploitation de la TI) donne le contrôle de la machine au programme au moment où il le lance, il faudra que celui-ci repasse la main au TIOS !
Pour cela, notre programme doit se terminer par cette instruction :

```
  rts
```

Cette instruction signifie "ReTurn from Subroutine". Elle signifie quitter la routine courante. Ici, la routine courante est notre programme ; la quitter signifie donc rendre le contrôle au TIOS.
Nous verrons un usage plus général de cette instruction lorsque nous travaillerons sur l'utilisation de fonctions, puisqu'elle nous permettra de quitter une fonction pour revenir à l'appellant...

C: Résumé

Voila, nous avons terminé l'écriture de notre programme. Le voici, écrit en un seul bloc, pour faciliter la lecture :

```
; Assembly Source File
; Created 21/09/2002, 19:02:03

    include "OS.h"
    xdef    _nostub
    xdef    _ti89
    xdef    _ti92plus

_main:
    ; Lors de nos prochains programmes, c'est ici que nous insérerons notre
code.
    rts
```

Exemple Complet

A présent, il nous faut lancer l'Assemblage de ce programme. Pour cela, cliquez sur "Project", puis "Build", ou alors, enfoncez la touche F9 de votre clavier.
Une fois l'assemblage terminé, vous pouvez tester votre programme. Pour ce faire, je vous conseille de toujours utiliser un émulateur (tel VTI, disponible sur http://www.ticalc.org), et de n'envoyer le programme sur votre vraie calculatrice que lorsque vous aurez pu constater qu'il était, à première vue du moins, dépourvu de Bug. Souvenez-vous en pour la suite de ce tutorial, ainsi que pour les programmes que vous serez amené à développer par vous-même, mais aussi, par prudence, pour ceux que vous pouvez télécharger sur Internet (On n'est jamais trop prudent) !